



TITLE:

# Bottom-Left安定点の効率的な列挙法とその応用 (最適化モデルとアルゴリズムの新展開)

AUTHOR(S):

Imahori, Shinji; Chien, Yuyao; Tanaka, Yuma;  
Yagiura, Mutsunori

---

CITATION:

Imahori, Shinji ...[et al]. Bottom-Left安定点の効率的な列挙法とその応用 (最適化モデルとアルゴリズムの新展開). 数理解析研究所講究録 2011, 1726: 39-49

ISSUE DATE:

2011-02

URL:

<http://hdl.handle.net/2433/170510>

RIGHT:

## Bottom-Left 安定点の効率的な列挙法とその応用

名古屋大学大学院・工学研究科 今堀 慎治 (Shinji Imahori)  
Graduate School of Engineering,  
Nagoya University

名古屋大学大学院・情報科学研究科 簡 于耀 (Yuyao Chien)  
田中 勇真 (Yuma Tanaka)  
柳浦 睦憲 (Mutsunori Yagiura)  
Graduate School of Information Science,  
Nagoya University

概要: 本稿では, 既に配置された長方形の集合と 1 つの未配置の長方形が与えられたとき, 未配置長方形の Bottom-Left 安定点を全て列挙する問題を考える. 提案アルゴリズムは, 配置された長方形の数を  $n$ , 未配置の長方形の Bottom-Left 安定点の数を  $K$  とすると,  $O((n+K)\log n)$  時間で全ての Bottom-Left 安定点を列挙する. 提案アルゴリズムの特徴の一つに, 既配置長方形が Bottom-Left 安定でない場合や, 既配置長方形間に重なりがある場合にも適用可能であることが挙げられる. 計算量の理論的な解析と, 数値実験による評価を通して, 提案アルゴリズムの有効性を検証する.

### 1 Introduction

Rectangle packing is an important problem with applications in steel, wood, glass, paper and many other industries. There are a number of variants of the problem with different objectives and constraints, but the essential task is to place a given set of rectangles in a given larger area without overlap so that the wasted space in the resulting layout is minimized. (See [14] for a typology of cutting and packing problems.) Almost all variants of the problem are known to be NP-hard, and many heuristic algorithms have been proposed in the literature. One of the typical frameworks of existing heuristic algorithms is the *bottom-left strategy*, which places rectangles one by one at *bottom-left stable positions* [1, 9, 12]. A fundamental problem to be solved for executing these algorithms is to enumerate all bottom-left stable positions (or to find a bottom-left stable position with some properties) for a set of already placed rectangles and one new rectangle to be placed next.

Bottom-left stable positions are defined for a given area (in this paper, we assume the shape of the given area is rectangular), a set of rectangles placed in the area, and one new rectangle. A bottom-left stable position is a point in the area where the new rectangle can be placed without overlap with already placed rectangles and the new rectangle cannot move to the bottom or to the left. We note that there are many bottom-left stable positions in general and the lowest one (if there are ties, the leftmost one among the lowest) is called the *bottom-left position*. We also define *bottom-left stability* for a layout; if there is no overlap among rectangles and no rectangle can move to the bottom or to the left, the layout satisfies bottom-left stability.

Some constructive heuristic algorithms for the rectangle packing problem place rectangles at a bottom-left stable position [1, 3, 7, 9, 12], and hence any layouts constructed by these algorithms (including intermediate layouts) satisfy bottom-left stability. For layouts with bottom-left stability, Chazelle [4] showed that the number of bottom-left stable positions for a new rectangle is at most  $n+1$  when the number of placed rectangles is  $n$  and proposed an algorithm to enumerate all the bottom-left stable positions in linear time.

Another common framework of heuristic algorithms for the rectangle packing problem is improvement method, which places all the rectangles in the given area without overlap and iteratively improves the layout by some operations. This kind of algorithms often place a rectangle at a bottom-left stable position, but in this case, they may need to solve the problem of finding such a position in a layout *without bottom-left stability*. For this case, Healy et al. [6] showed that the number of bottom-left stable positions for a new rectangle is  $O(n)$  when  $n$  rectangles are placed in the area without overlap, and they proposed an  $O(n \log n)$  time algorithm to enumerate all the bottom-left stable positions.

For some packing problems including the two-dimensional irregular packing problem, algorithms with *compaction and separation operations* were proposed [2, 5, 8, 11, 13]. These algorithms generate layouts with overlap during their execution. However, efficient algorithms to enumerate bottom-left stable positions in layouts with overlap have not been proposed yet.

In this paper, we consider the problem of enumerating bottom-left stable positions for a new rectangle within a given layout that may not satisfy bottom-left stability and may have overlap between rectangles. We propose an enumeration algorithm that runs in  $O((n+K) \log n)$  time, where  $n$  is the number of placed rectangles and  $K$  is the number of bottom-left stable positions. Our algorithm enumerates bottom-left stable positions from bottom to top (from left to right for positions with same  $y$ -coordinates), and hence it outputs the bottom-left position first in  $O(n \log n)$  time.

The bottom-left strategy can naturally be generalized to the three-dimensional case. An important consequence of the algorithm proposed in this paper is that it can be utilized to design an efficient algorithm to execute such a bottom-left algorithm for the three-dimensional packing problem. Kawashima et al. [10] showed that the time complexity was improved from the previous best-known  $O(n^5)$  to  $O(n^3 \log n)$ . In their proof, our algorithm is used as a core part of the algorithm, and the applicability of our algorithm to the case with rectangles having overlap is crucial, i.e., existing algorithms such as those proposed in [4, 6] cannot be used for this purpose.

## 2 Problem description

We are given a set of  $n$  rectangles  $I = \{1, 2, \dots, n\}$  and one large rectangular area, also called the container. The container has its width and height  $(W, H)$  and its bottom left point is placed at  $(0, 0)$  in the plane. Each rectangle  $i \in I$  has its width and height  $(w_i, h_i)$ , and is placed orthogonally in the plane. Let  $(x_i, y_i)$  be the coordinate of the bottom left point of rectangle  $i$ . We note that the given rectangles may protrude from the container and may have overlap each other. We are also given one new rectangle  $j \notin I$  with size  $(w_j, h_j)$  that has not been placed yet. The objective is to enumerate all the bottom-left stable positions in the container for rectangle  $j$ .

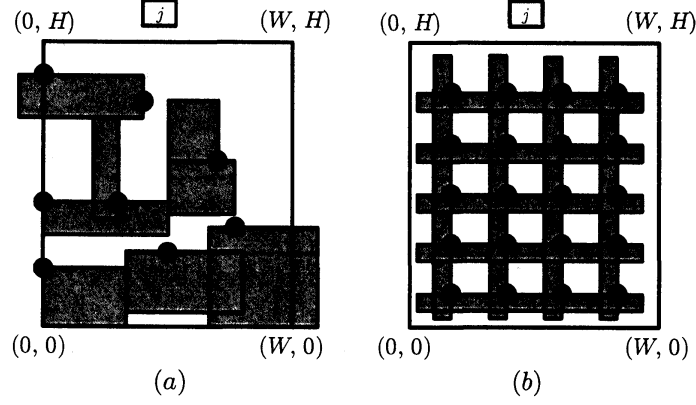


Figure 1: Bottom-left stable positions.

See Figure 1(a) for an example of bottom-left stable positions; black points in this figure denote bottom-left stable positions for rectangle  $j$ . Let  $K$  be the number of bottom-left stable positions for a given layout and one new rectangle. It is known that  $K = O(n^2)$  and  $K$  can be  $\Theta(n^2)$  for some cases (see Figure 1(b) for an example).

### 3 Algorithms

In this section, we propose algorithms to enumerate bottom-left stable positions. We first introduce *no-fit polygon*, which is often used in packing algorithms to check overlap efficiently. In Section 3.2, we give a technique to compute for each point  $p$  in the plane the number of no-fit polygons containing  $p$  by using *sweep line*. In Section 3.3, we propose an algorithm for enumerating bottom-left stable positions. We estimate the computational complexity of our algorithms in Section 3.4.

Instead of considering the constraint that requires a new rectangle to be placed in the container, we use a set of four sufficiently large virtual rectangles  $C = \{c_l, c_r, c_t, c_b\}$  that satisfies the following condition: Rectangle  $j$  does not have overlap with rectangles  $i' \in I \cup C$  if and only if it is placed in the container without overlap with rectangles  $i \in I$ . We call these virtual rectangles container rectangles; see Figure 2 for an example of container rectangles. We denote  $I' = I \cup C$ ; then  $|I'| = |I| + 4$  holds.

#### 3.1 No-fit polygon

*No-fit polygon* (NFP) is a geometric technique to check overlap of two polygons in two-dimensional space. It is defined for an ordered pair of two polygons  $i$  and  $j$ , where the position of polygon  $i$  is fixed and polygon  $j$  can be moved.  $NFP(i, j)$  denotes positions of polygon  $j$  having intersection with polygon  $i$ . In this paper, we treat only the situation that both of two polygons  $i$  and  $j$  are rectangles, and hence it is easy to compute  $NFP(i, j)$ . Let rectangle  $i$  be placed at  $(x_i, y_i)$  and rectangle  $j$  be the new rectangle. Then  $NFP(i, j)$  is defined as follows:

$$NFP(i, j) = \{(x, y) \mid x_i - w_j < x < x_i + w_i, y_i - h_j < y < y_i + h_i\}.$$

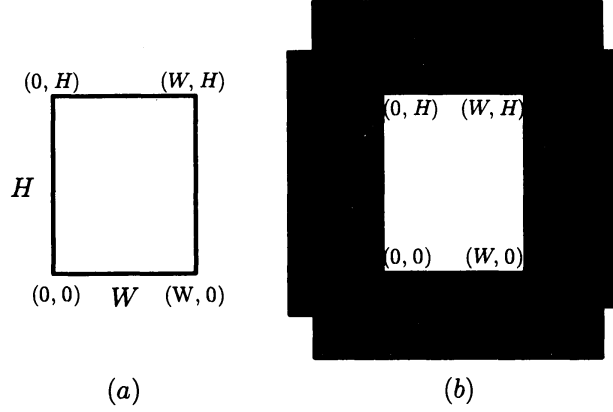


Figure 2: (a) The given area, (b) Container rectangles to represent the area.

We also define the *overlap number*  $B(x, y)$  of no-fit polygons at point  $(x, y)$  as follows:

$$B(x, y) = |\{i \in I' \mid (x, y) \in NFP(i, j)\}|.$$

By using this overlap number, we can characterize bottom-left stable positions as follows:

$$\begin{aligned} (x, y) \text{ is a bottom-left stable position} &\iff \\ B(x, y) = 0 \wedge B(x - \varepsilon, y) > 0 \wedge B(x, y - \varepsilon) > 0, \end{aligned}$$

where  $\varepsilon$  is any sufficiently small positive number. In the next section, we will describe how to compute overlap number of no-fit polygons.

### 3.2 Compute overlap numbers

The algorithm first computes all no-fit polygons  $NFP(i, j)$  of rectangle  $j$  relative to placed and container rectangles  $i \in I'$ . In order to compute overlap numbers (of no-fit polygons) in the given area efficiently, the algorithm uses a sweep line parallel to the  $x$ -axis and moves it from bottom to top.

Let  $N_t$  (resp.,  $N_b$ ) be the set of all the top (resp., bottom) edges of no-fit polygons and  $N_{tb} = N_t \cup N_b$ . The overlap numbers on the sweep line will be changed only when the sweep line encounters a member of  $N_{tb}$ , and changes occur only in the interval between the left edge and right edge of the no-fit polygon encountered by the sweep line.

Let  $N_l$  (resp.,  $N_r$ ) be the set of all the left (resp., right) edges of no-fit polygons and  $N_{lr} = N_l \cup N_r$ . Because there are  $n$  placed rectangles and four container rectangles,  $|N_t| = |N_b| = |N_l| = |N_r| = n + 4$  and  $|N_{tb}| = |N_{lr}| = 2n + 8$  hold. The elements in  $N_{lr}$  are sorted in nondecreasing order of the  $x$ -coordinates of the elements, where ties are broken by putting more priority to elements in  $N_r$ . This tie-breaking rule is important, because if two no-fit polygons have their left and right boundaries at the same  $x$ -coordinate, the new rectangle  $j$  can be placed without overlap at the intersection point of boundaries of the two no-fit polygons. Let  $x_{lr}^{(k)}$  be the  $x$ -coordinate of the  $k$ th element in the sorted list of  $N_{lr}$ , and define intervals

$$S_k = [x_{lr}^{(k)}, x_{lr}^{(k+1)}], \quad k = 1, 2, \dots, 2n + 7$$

on the sweep line.

The algorithm maintains the overlap number for each interval  $S_k$  during the computation. Initially, the sweep line is at a sufficiently low position, and it overlaps with no NFP. At this moment, the overlap number of every interval  $S_k$  is zero.

We now consider the moment when the sweep line encounters a member in  $N_{tb}$ . Let  $NFP(i, j)$  be the rectangle whose top or bottom edge is encountered by the sweep line, and assume that the left (resp., right) edge of  $NFP(i, j)$  is the  $l$ th (resp.,  $(r+1)$ st) element in the sorted list of  $N_{lr}$ . In this situation, we should change the overlap numbers for intervals  $S_l, S_{l+1}, \dots, S_r$ . To be more precise, we should increase (resp., decrease) their overlap numbers by one if the encountered edge is a member of  $N_b$  (resp.,  $N_t$ ). To update overlap numbers on the sweep line efficiently, we use a complete binary tree whose leaves represent intervals  $S_1, S_2, \dots, S_{2n+7}$ . Here, the  $k$ th leaf from the left corresponds to the interval  $S_k$ , and the name of this leaf is  $k$ . We note that  $2n+7$  is not a power of two for any  $n$ , and there are remaining leaves on the right side of the leaf corresponding to interval  $S_{2n+7}$ . Such remaining leaves are called dummy leaves. We use a complete binary tree with the minimum number of dummy leaves. Then the number of dummy leaves is less than  $2n+7$  and the height of this tree is  $O(\log n)$ . Every node of this tree stores values  $p_{\text{self}}$ ,  $p_{\text{min}}$  and  $p_{\text{max}}$ , whose role will be explained later.

For two nodes  $u$  and  $v$  of the tree, let  $PATH(u, v)$  be the set of nodes in the path from  $u$  to  $v$  including  $u$  and  $v$  themselves. Let  $g(k)$  be the overlap number for interval  $S_k$  of the sweep line. (To be more precise,  $g(k)$  is the overlap number of all points in  $S_k$  except the left (resp., right) boundary of  $S_k$  if it corresponds to the left (resp., right) boundary of an NFP. Thus we need to treat the boundaries carefully considering that each NFP is an open set. When the value of  $y$  is fixed to the height of the current sweep line, the function  $B(x, y)$  is a lower semicontinuous piecewise linear function consisting of horizontal line segments with heights  $g(1), g(2), \dots, g(2n+7)$  aligned in this order from left to right.) For every dummy leaf  $u$ , we set  $g(u) = 1$ .

The algorithm maintains the values of  $p_{\text{self}}$  for all nodes of the tree so that

$$\sum_{u \in PATH(k, \text{root})} p_{\text{self}}(u) = g(k)$$

is satisfied for each leaf  $k$ , where  $\text{root}$  is the root node of the binary tree. Then it is possible to compute the overlap number of an interval in  $O(\log n)$  time by using the values of  $p_{\text{self}}$  in the path from the corresponding leaf to the root node. We also define the values of  $p_{\text{min}}(v)$  and  $p_{\text{max}}(v)$  for each node  $v$  of the complete binary tree as follows:

$$p_{\text{min}}(v) = \min_{k \in Q(v)} \sum_{u \in PATH(k, v)} p_{\text{self}}(u), \quad (1)$$

$$p_{\text{max}}(v) = \max_{k \in Q(v)} \sum_{u \in PATH(k, v)} p_{\text{self}}(u), \quad (2)$$

where  $Q(v)$  is the set of all leaf nodes in the subtree rooted at the node  $v$ . By using the value of  $p_{\text{min}}(v)$  (resp.,  $p_{\text{max}}(v)$ ) and the values of  $p_{\text{self}}(u)$  for nodes  $u$  in the path from the parent node of  $v$  to the root node, it is possible to check whether leaf nodes whose overlap numbers are

equal to zero (resp., positive) exist in  $Q(v)$ . Let  $u$  and  $u'$  be the children of node  $v$  and assume that the values of  $p_{\min}(u)$  and  $p_{\min}(u')$  are known. In this situation, the value of  $p_{\min}(v)$  can be computed in constant time by

$$p_{\min}(v) = p_{\text{self}}(v) + \min\{p_{\min}(u), p_{\min}(u')\}. \quad (3)$$

The value of  $p_{\max}(v)$  can be computed similarly.

We now explain the algorithm to keep the values of  $p_{\text{self}}$ ,  $p_{\min}$  and  $p_{\max}$  appropriately. Consider the moment when the sweep line encounters a member in  $N_{\text{tb}}$ . Let  $NFP(i, j)$  be the rectangle whose top or bottom edge is encountered by the sweep line, and let the left (resp., right) edge of  $NFP(i, j)$  be the  $l$ th (resp.,  $(r + 1)$ st) element in the sorted list of  $N_{\text{lr}}$ . Here we assume for simplicity that the encountered edge is a bottom edge of the NFP. The case when a top edge is encountered is similar; instead of increasing the values by one, the algorithm decreases the values by one. The algorithm first finds the leaves  $l$  and  $r$  that correspond to the  $l$ th and  $r$ th intervals and increases the values of  $p_{\text{self}}$ ,  $p_{\min}$  and  $p_{\max}$  of these leaf nodes by one. It then traverses nodes in the paths from the leaf nodes  $l$  and  $r$  to their least common ancestor  $v$ . During this traversal, whenever a node in the path from  $l$  (resp.,  $r$ ) to  $v$  is reached from its left (resp., right) child, the algorithm increases the values of  $p_{\text{self}}$ ,  $p_{\min}$  and  $p_{\max}$  of the right (resp., left) child by one. It also updates  $p_{\min}$  (by using (3)) and  $p_{\max}$  for nodes in the paths from  $l$  and  $r$  to  $v$  so that the conditions (1) and (2) are satisfied. Finally, the algorithm updates the values of  $p_{\min}$  and  $p_{\max}$  for all nodes in the path from  $v$  to the root node of the tree. The details of this procedure is summarized as Algorithm UPDATEVALUES( $\lambda, l, r$ ).

#### **Algorithm** UPDATEVALUES( $\lambda, l, r$ )

**Input:** An index  $\lambda$  ( $\lambda = 1$  when the sweep line encounters a bottom edge of NFP;  $\lambda = -1$  when the sweep line encounters a top edge of NFP), two leaves  $l$  and  $r$  corresponding to the leftmost and rightmost intervals and current values of  $p_{\text{self}}$ ,  $p_{\min}$  and  $p_{\max}$ .

**Task:** Update the values of  $p_{\text{self}}$ ,  $p_{\min}$  and  $p_{\max}$ .

**Step 1:** Increase the values of  $p_{\text{self}}(l), p_{\min}(l), p_{\max}(l), p_{\text{self}}(r), p_{\min}(r), p_{\max}(r)$  by  $\lambda$  (this means that if  $\lambda = -1$ , these values are actually decreased by one).

**Step 2:** Let  $l_{\text{prev}} := l$  and  $r_{\text{prev}} := r$ , and then let  $l$  be the parent of  $l$  and  $r$  be the parent of  $r$ . If  $l \neq r$ , proceed to Step 3; otherwise go to Step 4.

**Step 3:** If the right (resp., left) child  $u$  of  $l$  (resp.,  $r$ ) is different from  $l_{\text{prev}}$  (resp.,  $r_{\text{prev}}$ ), increase the values of  $p_{\text{self}}(u), p_{\min}(u)$  and  $p_{\max}(u)$  by  $\lambda$ . Let  $p_{\min}(l) := p_{\text{self}}(l) + \min\{p_{\min}(u), p_{\min}(u')\}$ , where  $u$  and  $u'$  are the children of node  $l$ . Update the values of  $p_{\max}(l), p_{\min}(r), p_{\max}(r)$  similarly. Return to Step 2.

**Step 4:** For each node  $v$  in the path from  $l(=r)$  to the root and the children  $u$  and  $u'$  of  $v$ , let  $p_{\min}(v) := p_{\text{self}}(v) + \min\{p_{\min}(u), p_{\min}(u')\}$  and update  $p_{\max}(v)$  similarly. Then stop.

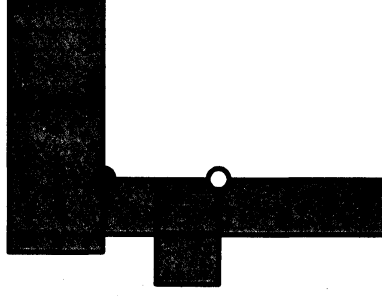


Figure 3: No-fit polygons whose top edges have the same  $y$ -coordinate.

### 3.3 Enumerate bottom-left stable positions

We explain our algorithm that enumerates bottom-left stable positions. Observe that, while the sweep line parallel to the  $x$ -axis is moved from bottom to top, the overlap numbers of no-fit polygons for intervals in the sweep line decrease only if the top edge of a no-fit polygon is encountered. This means that bottom-left stable positions can be found only in this case, because a point  $(x, y)$  can be a bottom-left stable position only if  $B(x, y) = 0$  and  $B(x, y - \varepsilon) > 0$  for any sufficiently small positive  $\varepsilon$ . For this reason, when the sweep line encounters the bottom edge of a no-fit polygon, the algorithm just updates the overlap numbers according to the rule described in Section 3.2. On the other hand, when the sweep line encounters the top edge of a no-fit polygon, the algorithm updates the overlap numbers and outputs bottom-left stable positions on the sweep line if such positions exist. To manage these events, the elements in  $N_{tb}$  are sorted in nondecreasing order of the  $y$ -coordinates of the elements, where ties are broken by putting more priority to elements in  $N_t$ . If the top edges of some no-fit polygons have the same  $y$ -coordinate, we put more priority to those elements that correspond to no-fit polygons whose left edge have smaller  $x$ -coordinates.

At any point  $(x, y)$  such that the overlap number  $B(x, y)$  is equal to zero, we can place rectangle  $j$  without overlap. Moreover, if the overlap number becomes zero when the top edge of a no-fit polygon is encountered by the sweep line, then  $B(x, y - \varepsilon) > 0$  for any sufficiently small  $\varepsilon > 0$ , i.e., rectangle  $j$  cannot move downward from the point. Furthermore, if the point  $(x, y)$  is at the left boundary of an interval  $S_k$  and its left adjacent interval  $S_{k-1}$  has a positive overlap number, then  $B(x - \varepsilon, y) > 0$  for any sufficiently small  $\varepsilon > 0$ , i.e., rectangle  $j$  cannot move to the left, provided that all the top edges of no-fit polygons whose  $y$ -coordinates are  $y$  have already been encountered by the sweep line. Such a point  $(x, y)$  is a bottom-left stable position, and our algorithm enumerates all such points. We note that, if there are some no-fit polygons whose top edges have the same  $y$ -coordinate, all such top edges should leave the sweep line together. Otherwise, the algorithm may output positions from which rectangle  $j$  can move to the left as depicted in Figure 3; when rectangle ‘ $i$ ’ leaves the sweep line, black and white points may be output as bottom-left stable positions. The details of our algorithm to enumerate bottom-left stable positions is summarized as Algorithm ENUMERATEBL( $j, I$ ).



**Algorithm** ENUMERATEBL( $j, I$ )

**Input:** Placed rectangles  $i \in I$  in the given area and one new rectangle  $j \notin I$ .

**Output:** All bottom-left stable positions for rectangle  $j$ .

- Step 1:** Compute all no-fit polygons of rectangle  $j$  relative to all placed and container rectangles  $i \in I'$ . Sort left and right edges  $N_{lr} := N_l \cup N_r$  of no-fit polygons with the rule described in Section 3.2. Create the minimum complete binary tree with at least  $2n+7$  leaves. Initialize the values  $p_{\text{self}}(u) := 0$ ,  $p_{\text{min}}(u) := 0$  and  $p_{\text{max}}(u) := 0$  for all leaf nodes  $u$  corresponding to intervals  $S_1$  to  $S_{2n+7}$ , and initialize the values  $p_{\text{self}}(u) := 1$ ,  $p_{\text{min}}(u) := 1$  and  $p_{\text{max}}(u) := 1$  for all dummy leaves  $u$ . For all internal nodes  $u$ , set  $p_{\text{self}}(u) := 0$  and compute the values of  $p_{\text{min}}(u)$  and  $p_{\text{max}}(u)$ . Sort top and bottom edges  $N_{tb} := N_t \cup N_b$  of no-fit polygons with the rule described in Section 3.3.
- Step 2:** Choose the first element  $e \in N_{tb}$  and let  $N_{tb} := N_{tb} \setminus \{e\}$ . If the  $y$ -coordinate of element  $e$  is greater than  $H - h_j$ , then stop.
- Step 3:** Let  $i \in I'$  be the no-fit polygon having the element  $e$  as its top or bottom edge, and assume that its left (resp., right) edge is the  $l$ th (resp.,  $(r+1)$ st) element in  $N_{lr}$ . If  $e \in N_t$  (resp.,  $N_b$ ), then set  $\lambda := -1$  (resp.,  $\lambda := 1$ ). Call algorithm UPDATEVALUES( $\lambda, l, r$ ).
- Step 4:** If  $e \in N_b$ , return to Step 2; otherwise (i.e.,  $e \in N_t$ ), set  $\alpha := l$  and  $\beta := r$ .
- Step 5:** Let  $e'$  be the first element in  $N_{tb}$ . If  $e$  and  $e'$  have the same  $y$ -coordinate and  $e' \in N_t$ , go to Step 6; otherwise go to Step 7.
- Step 6:** Let  $i' \in I'$  be the no-fit polygon having the element  $e'$  as its top edge, and assume that its left (resp., right) edge is the  $l'$ th (resp.,  $(r'+1)$ st) element in  $N_{lr}$ . If  $l' > \beta$  holds, go to Step 7; otherwise let  $N_{tb} := N_{tb} \setminus \{e'\}$  and call algorithm UPDATEVALUES( $-1, l', r'$ ). If  $r' > \beta$  holds, set  $\beta := r'$ . Return to Step 5.
- Step 7:** If the overlap number of interval  $S_\alpha$  is positive, go to Step 8. If the overlap number of interval  $S_\alpha$  is equal to 0 and that of interval  $S_{\alpha-1}$  is positive, then output a bottom-left stable position  $(x, y)$ , where  $x = x_{lr}^{(\alpha)}$  and  $y$  is the  $y$ -coordinate of the current sweep line. Go to Step 9.
- Step 8:** By using the values of  $p_{\text{min}}$ , find the leftmost interval  $S_\gamma$  that satisfies  $\gamma > \alpha$  and whose overlap number is equal to 0. If  $\gamma > \beta$  or such a  $\gamma$  is not found, return to Step 2; otherwise, output a bottom-left stable position  $(x, y)$ , where  $x = x_{lr}^{(\gamma)}$  and  $y$  is the  $y$ -coordinate of the current sweep line. Let  $\alpha := \gamma$  and go to Step 9.
- Step 9:** By using the values of  $p_{\text{max}}$ , find the leftmost interval  $S_\gamma$  that satisfies  $\gamma > \alpha$  and whose overlap number is positive. If  $\gamma \geq \beta$  or such a  $\gamma$  is not found, return to Step 2; otherwise, let  $\alpha := \gamma$  and go to Step 8.

The overlap number of every dummy leaf is initialized to one and is not changed during the execution of the algorithm, and hence the dummy leaf nodes have no influence on the output of

this algorithm. By executing this algorithm, all the bottom-left stable positions are enumerated from bottom to top (from left to right for positions with same  $y$ -coordinates).

In Step 8,  $\gamma$  is found as follows. The algorithm first climbs the binary tree from the leaf  $\alpha$ , and whenever a node  $v$  is reached from its left child, it checks whether the subtree rooted at the right child  $u$  of  $v$  has a leaf whose overlap number is zero. When the first node  $u$  having such a leaf is found, the algorithm goes down the tree from the  $u$ , choosing the left-most child including such a leaf. In Step 9, the leaf  $\gamma$  is found similarly.

### 3.4 Computational complexity

We estimate the time complexity of our algorithms described in Sections 3.2 and 3.3. Algorithm  $\text{UPDATEVALUES}(\lambda, l, r)$  runs in  $O(\log n)$  time since the height of the complete binary tree is  $O(\log n)$ . Algorithm  $\text{ENUMERATEBL}(j, I)$  calls algorithm  $\text{UPDATEVALUES}(\lambda, l, r)$  as a subroutine at most  $2n + 8$  times in Steps 3 and 6; then the total time for this part is  $O(n \log n)$ . The time complexity of Step 7 is  $O(\log n)$  and it is called at most  $n + 4$  times. The time complexity of Step 8 is  $O(\log n)$  because  $O(\log n)$  nodes are visited during the traversal from  $\alpha$  to  $\gamma$ , and it is possible for each node  $u$  to check whether the subtree rooted at the node  $u$  has a leaf node whose overlap number is equal to zero in constant time (this is not hard to see from the property explained in Section 3.2 just after equation (2)). The time complexity of Step 9 is also  $O(\log n)$  for a similar reason. Steps 8 and 9 are called at most  $n + 4 + K$  times, where  $K$  is the number of bottom-left stable positions enumerated. In summary, the total time complexity of algorithm  $\text{ENUMERATEBL}(j, I)$  is  $O((n + K) \log n)$ . Furthermore, if our algorithms are utilized for finding the bottom-left position (i.e., the lowest (if there are ties, the leftmost among ties) bottom-left stable position), algorithm  $\text{ENUMERATEBL}(j, I)$  can be stopped immediately when it finds the first bottom-left stable position. In this case, the time complexity to find the bottom-left position is  $O(n \log n)$ .

## 4 Computational results

In this section, we evaluate the proposed algorithms via computational experiments. All the algorithms were coded in C and experiments were conducted on a PC (Intel Xeon 3GHz, 2MB cache, 1GB memory). As a set of placed rectangles, we used test instances for the rectangle packing problem with 16, 32, 64,  $\dots$ , 1048576 rectangles (these instances are obtained electronically from <http://www.na.cse.nagoya-u.ac.jp/~imahori/packing/instance.html>). The size of a new rectangle to be placed next is similar to the placed rectangles.

Table 1 shows the computational results. Column “ $n$ ” shows the number of rectangles placed in a rectangular area. The number of bottom-left stable positions enumerated is shown in column “BLpts,” and computation time in seconds is reported in column “time(sec).” We note that layouts were randomly generated many times (at least 10 times; it depends on the number of placed rectangles) and the average number of bottom-left stable positions and average computation time were reported. From the table, we can observe that the proposed algorithm enumerates bottom-left stable positions in a short time. It can enumerate all the bottom-left stable positions among hundreds of rectangles within 0.001 seconds, it spends less than 0.1

Table 1: Performance of our algorithm.

$n$	BLpts	time(sec.)
16	7	$6.61 \times 10^{-6}$
32	9	$1.71 \times 10^{-5}$
64	15	$4.55 \times 10^{-5}$
128	39	$1.18 \times 10^{-4}$
256	76	$2.63 \times 10^{-4}$
512	138	$5.84 \times 10^{-4}$
1024	266	$1.25 \times 10^{-3}$
2048	504	$2.66 \times 10^{-3}$
4096	636	$5.72 \times 10^{-3}$
8192	1248	$1.26 \times 10^{-2}$
16,384	2375	$2.80 \times 10^{-2}$
32,768	2931	$6.97 \times 10^{-2}$
65,536	6044	$2.15 \times 10^{-1}$
131,072	6446	$6.54 \times 10^{-1}$
262,144	12,901	$1.73 \times 10^0$
524,288	16,036	$4.29 \times 10^0$
1,048,576	33,483	$1.04 \times 10^1$

seconds for instances with up to 32,768 rectangles, and it takes 10.4 seconds to enumerate all the bottom-left stable positions in layouts with about one million rectangles.

## 5 Conclusions

In this paper, the problem of enumerating bottom-left stable positions for a given layout of rectangles was studied. We proposed an algorithm that enumerates all the bottom-left stable positions in  $O((n + K) \log n)$  time, where  $n$  is the number of placed rectangles and  $K$  is the number of bottom-left stable positions (i.e., size of the output). Our algorithm works for layouts without bottom-left stability and with overlaps. We also evaluated the proposed algorithm via computational experiments. Even for instances with more than one million placed rectangles, the proposed algorithm works in a short computation time.

A direction of future work is to propose faster algorithms: Our algorithm runs in  $O((n + K) \log n)$  time, but the existence of algorithms that run in  $O(n \log n + K)$  is open.

## References

- [1] Baker, B. S., Coffman, E. G., Rivest, R. L., Orthogonal packings in two dimensions. SIAM Journal on Computing. 9 (1980) 846–855.
- [2] Bennell, J. A., Dowsland, K.A., Hybridising tabu search with optimisation techniques for irregular stock cutting. Management Science. 47 (2001) 1160–1172.

- [3] Burke, E. K., Kendall, G., Whitwell, G., A new placement heuristic for the orthogonal stock-cutting problem. *Operations Research*. 52 (2004) 655–671.
- [4] Chazelle, B., The bottom-left bin packing heuristic: an efficient implementation. *IEEE Transactions on Computers*. 32 (1983) 697–707.
- [5] Gomes, A. M., Oliveira, J. F., Solving irregular strip packing problems by hybridising simulated annealing and linear programming. *European Journal of Operational Research*. 171 (2006) 811–829.
- [6] Healy, P., Creavin, M., Kuusik, A., An optimal algorithm for rectangle placement. *Operations Research Letters*. 24 (1999) 73–80.
- [7] Imahori, S., Yagiura, M., The best-fit heuristic for the rectangular strip packing problem: an efficient implementation and the worst-case approximation ratio. *Computers and Operations Research*. 37 (2010) 325–333.
- [8] Imamichi, T., Yagiura, M., Nagamochi, H., An iterated local search algorithm based on nonlinear programming for the irregular strip packing problem. *Discrete Optimization*. 6 (2009) 345–361.
- [9] Jakobs, S., On genetic algorithms for the packing of polygons. *European Journal of Operational Research*. 88 (1996) 165–181.
- [10] Kawashima, H., Tanaka, Y., Imahori, S., Yagiura, M., An efficient implementation of a constructive algorithm for the three-dimensional packing problem (in Japanese). *Proceedings of the 9th Forum on Information Technology 2010 (FIT2010)*, Issue 1, pp. 31–38.
- [11] Li, Z., Milenkovic, V., Compaction and separation algorithms for non-convex polygons and their applications. *European Journal of Operational Research*. 84 (1995) 539–561.
- [12] Liu, D., Teng, H., An improved BL-algorithm for genetic algorithm of the orthogonal packing of rectangles. *European Journal of Operational Research*. 112 (1999) 413–420.
- [13] Umetani, S., Yagiura, M., Imahori, S., Imamichi, T., Nonobe, K., Ibaraki, T., Solving the irregular strip packing problem via guided local search for overlap minimization. *International Transactions in Operational Research*. 16 (2009) 661–683.
- [14] Wäscher, G., Haussner, H., Schumann, H., An improved typology of cutting and packing problems. *European Journal of Operational Research*. 183 (2007) 1109–1130.